Data Mining

선형 대수 (Linear Algebra)

학습 목표

- 다차원 데이터를 표현하기 위한 벡터와 행렬을 리스트를 이용해서 정의해보고 관련 연산을 직접 구현해본다.

주요 내용

1. 벡터
2. 행렬

1. 벡터

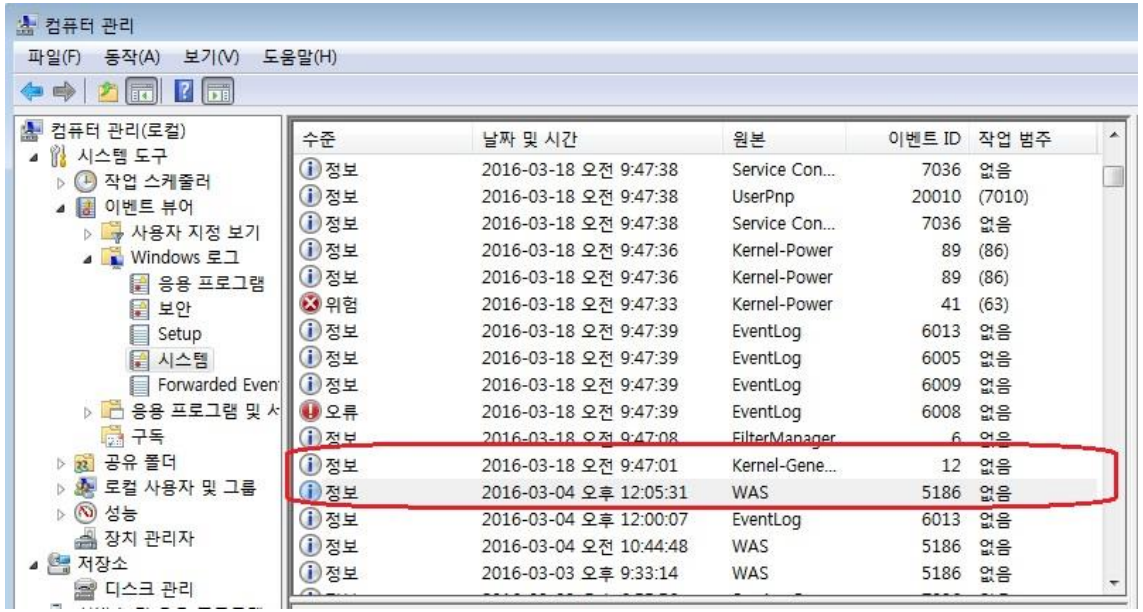
다차원 데이터

실생활에서의 데이터는 대부분 다차원으로 구성되어 있다.

사람에 대한 특징을 표현한다면?

(키, 몸무게, 나이, 성별, 학력, 직업, ...)

시스템 이벤트 정보



(수준, 날짜 및 시간, 원본, 이벤트 ID, 작업 범주...)

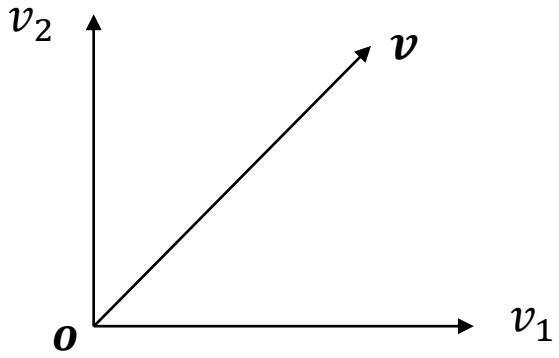
벡터 (Vector)

다차원 공간의 한 점은 벡터로 표현할 수 있다.

벡터 (Vector)

- 벡터 공간의 한 점
- 방향과 길이로 표현됨

$$\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$$



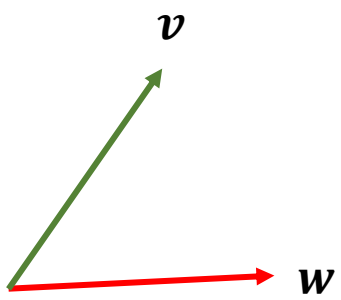
벡터

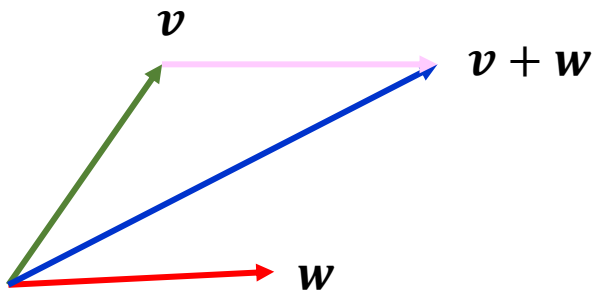
```
from typing import List

Vector = List[float]
```

- 벡터를 실수의 리스트로 정의

벡터 더하기

$\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$
$\boldsymbol{w} = [w_1, w_2, \ldots, w_n]$





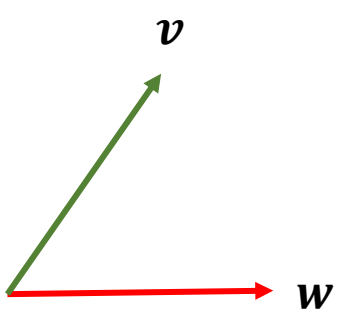
$$\boldsymbol{v} + \boldsymbol{w} = [v_1 + w_1, v_2 + w_2, \quad \ldots, v_n + w_n]$$

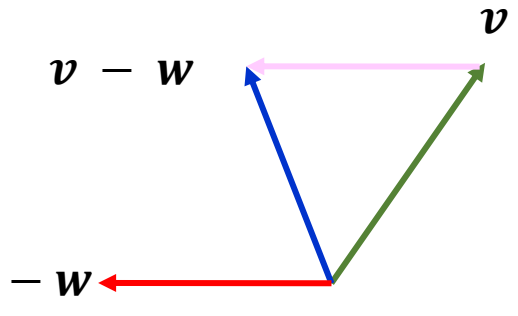
벡터 더하기

```python
def add(v: Vector, w: Vector) -> Vector:
    """Adds corresponding elements"""
    assert len(v) == len(w), "vectors must be the same length"

    return [v_i + w_i for v_i, w_i in zip(v, w)]
```

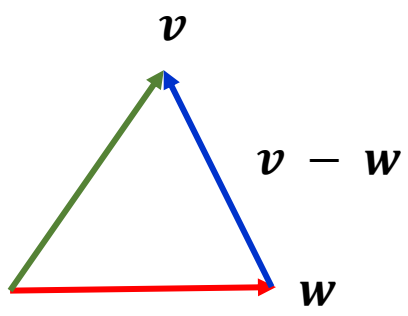
벡터 빼기

$\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$
$\boldsymbol{w} = [w_1, w_2, \ldots, w_n]$





or



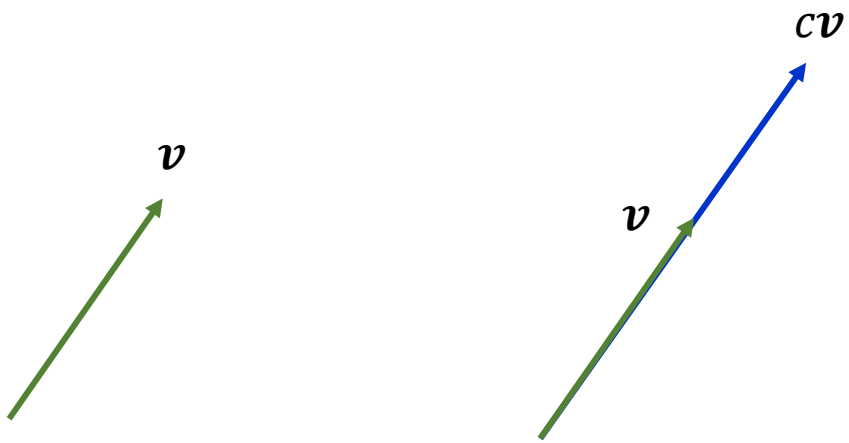
$$\boldsymbol{v} - \boldsymbol{w} = [v_1 - w_1, v_2 - w_2, \quad \ldots, v_n - w_n]$$

벡터 빼기

```python
def subtract(v: Vector, w: Vector) -> Vector:
    """Subtracts corresponding elements"""
    assert len(v) == len(w), "vectors must be the same length"

    return [v_i - w_i for v_i, w_i in zip(v, w)]
```

스칼라 곱

$$\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$$



$$c\boldsymbol{v} = [cv_1, cv_2, \ldots, cv_n]$$

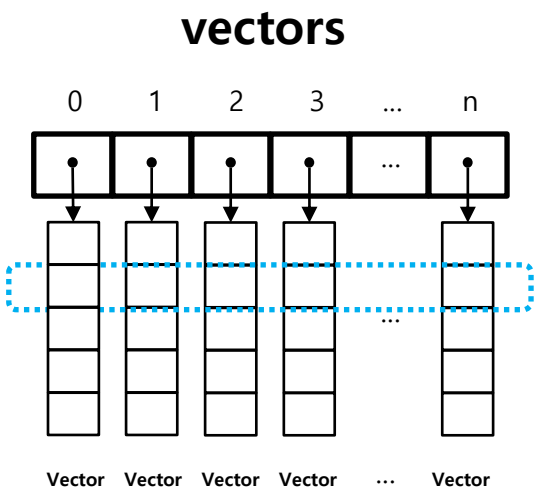
스칼라 곱

```
def scalar_multiply(c: float, v: Vector) -> Vector:
    """Multiplies every element by c"""
    return [c * v_i for v_i in v]
```

다중 벡터

여러 벡터 더하기

```python
def vector_sum(vectors: List[Vector]) -> Vector:
    """Sums all corresponding elements"""
    # Check that vectors is not empty
    assert vectors, "no vectors provided!"

    # Check the vectors are all the same size
    num_elements = len(vectors[0])
    assert all(len(v) == num_elements for v in vectors), "different sizes!“

    # the i-th element of the result is the sum of every vector[i]
    return [sum(vector[i] for vector in vectors)
            for i in range(num_elements)]
```

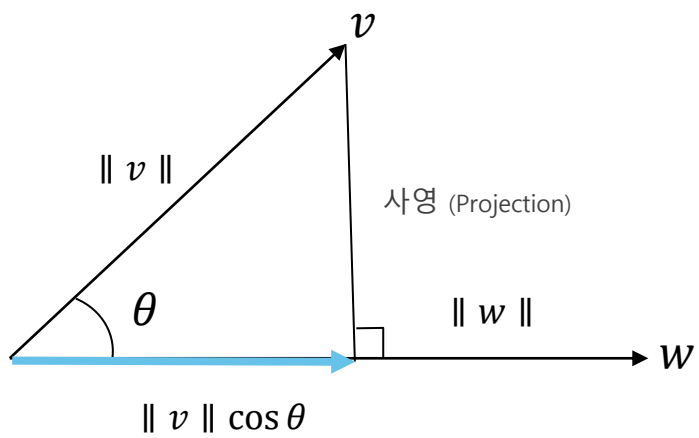


- 모든 벡터의 크기가 같은 지 확인
- 각 차원 i 별로 벡터의 요소들을 합산

여러 벡터 평균 구하기

```python
def vector_mean(vectors: List[Vector]) -> Vector:
    """Computes the element-wise average"""
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))
```

내적 (dot product)



내적 : $v \cdot w = \| v \| \| w \| \cos\theta$

$= \| v \| \cos\theta \| w \|$

사영

- 내적은 사영에 비례하기 때문에 사영 대신 내적을 사용하기도 함

내적 (Dot product)

- 두 벡터 v와 w가 이루는 각도가 θ일 때

  $v \cdot w = \| v \| \| w \| \cos\theta$ 를 내적이라고 함

$$v \cdot w = \sum_{i=0}^{n} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_n w_n$$

사영 (Projection)

$$\text{사영} = \frac{v \cdot w}{\| w \|} = \| v \| \cos\theta$$

내적 (dot product)

$$v \cdot w = \sum_{i=0}^{n} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_n w_n$$

내적

```
def dot(v: Vector, w: Vector) -> float:
    """Computes v_1 * w_1 + ... + v_n * w_n"""
    assert len(v) == len(w), "vectors must be same length"

    return sum(v_i * w_i for v_i, w_i in zip(v, w))
```

노름과 거리 함수

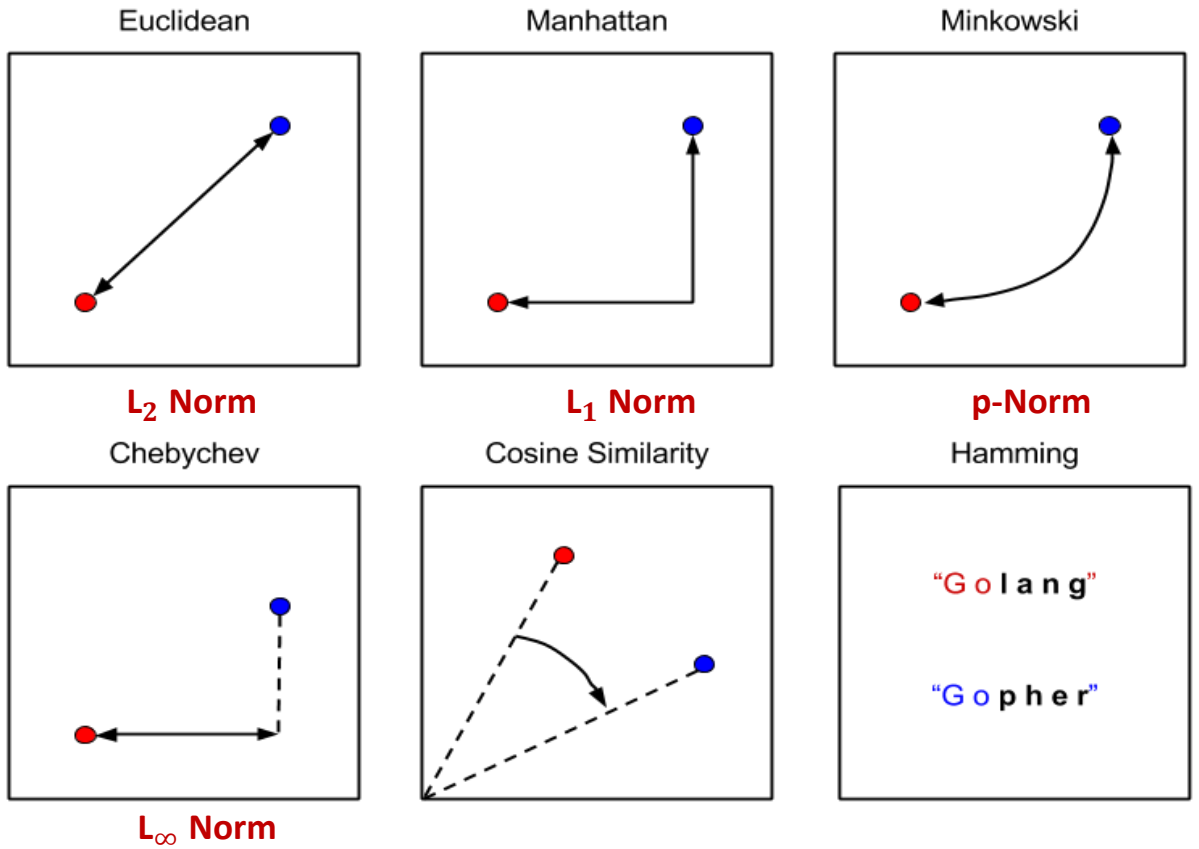
벡터의 크기 노름 (Norm)

p-노름 (p-Norm)

$$\| x \|_p = \left(\sum_{i=0}^{n} |x_i|^p \right)^{1/p} \quad \text{for } P \geq 1$$

- p=1 : L_1 Norm (Manhattan)
- p=2 : L_2 Norm (Euclidean)
- p=∞ : L_∞ Norm (Chebychev)

거리 (Distance) 함수의 종류



노름 (Norm)

L_2 노름 $\| v \|_2 = \sqrt{\sum_{i=0}^{n} |v_i|^2}$ $\| v \|^2 = \langle v, v \rangle$

제곱합

제곱합 (Sum of Square)

```
def sum_of_squares(v: Vector) -> float:
    """Returns v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)
```

L_2 노름

```
import math

def magnitude(v: Vector) -> float:
    """Returns the magnitude (or length) of v"""
    return math.sqrt(sum_of_squares(v))   # math.sqrt is square root function
```

거리 (distance)

유클리드 거리 (Euclidean distance) : $d(\boldsymbol{v}, \boldsymbol{w}) = \| \boldsymbol{v} - \boldsymbol{w} \|_2 = \sqrt{\sum_{i=0}^{n} |\boldsymbol{v}_i - \boldsymbol{w}_i|^2}$

거리의 제곱

유클리드 거리

```python
def distance(v: Vector, w: Vector) -> float:
    """Computes the distance between v and w"""
    return math.sqrt(squared_distance(v, w))
```

거리의 제곱

```python
def squared_distance(v: Vector, w: Vector) -> float:
    """Computes (v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
    return sum_of_squares(subtract(v, w))
```

2. 행렬

행렬 (Matrix)

행렬은 다음과 같은 용도로 사용할 수 있다.

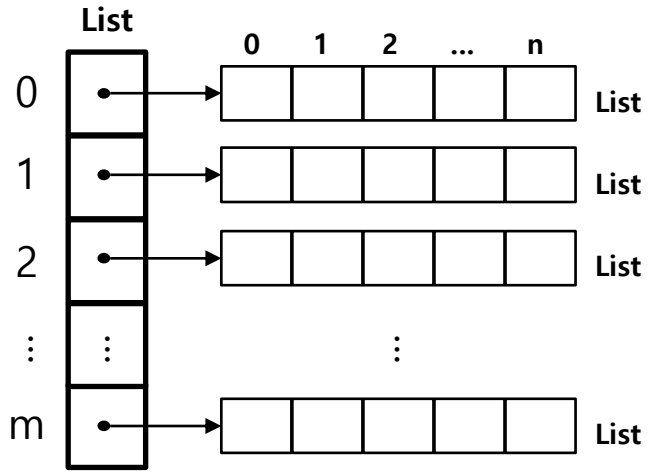
1. **데이터셋**
2. **선형 변환 연산**
3. **그래프의 인접 행렬**

$$A_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \ddots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

행렬 데이터 구조

행렬은 리스트의 리스트 (List of List)**로 표현**

```
Matrix = List[List[float]]
```



2x3 행렬 예시

```
A = [[1, 2, 3],
     [4, 5, 6]]
```

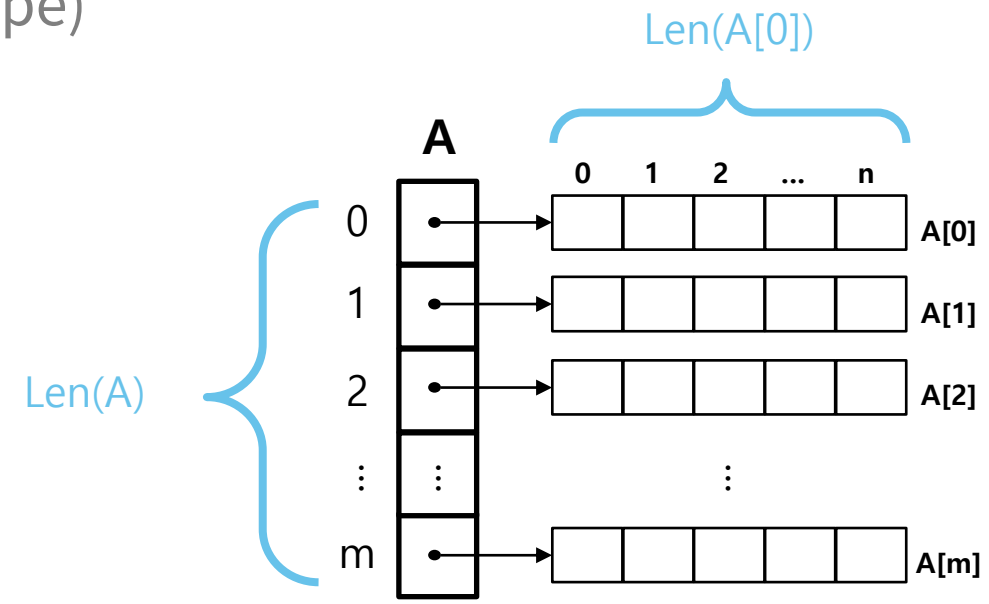
1	2	3
4	5	6

3x2 행렬 예시

```
B = [[1, 2],
     [3, 4],
     [5, 6]]
```

1	2
3	4
5	6

형태 (shape)



shape(A) = (len(A), len(A[0]))

행렬의 형태

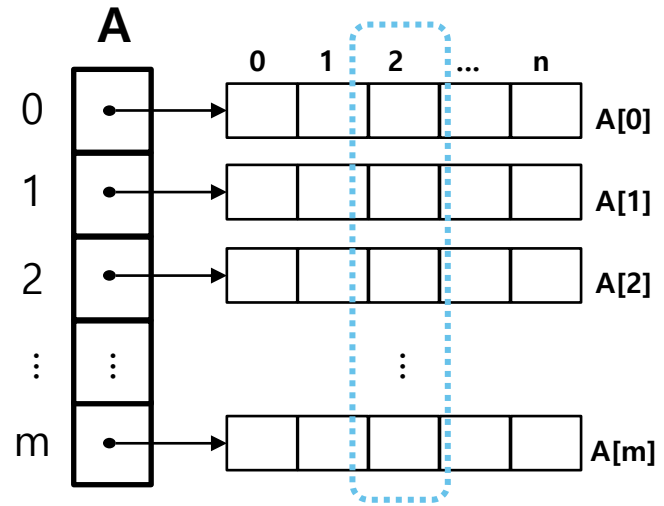
```python
from typing import Tuple

def shape(A: Matrix) -> Tuple[int, int]:
    """Returns (# of rows of A, # of columns of A)"""
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0   # number of elements in first row
    return num_rows, num_cols
```

- num_rows는 행렬 A의 길이
- num_cols는 행렬 A의 첫번째 row인 A[0]의 길이

행과 열

$$\text{col} = [A[0][j], A[1][j], \ldots, A[m][j]]$$

$$\text{row} = A[i]$$



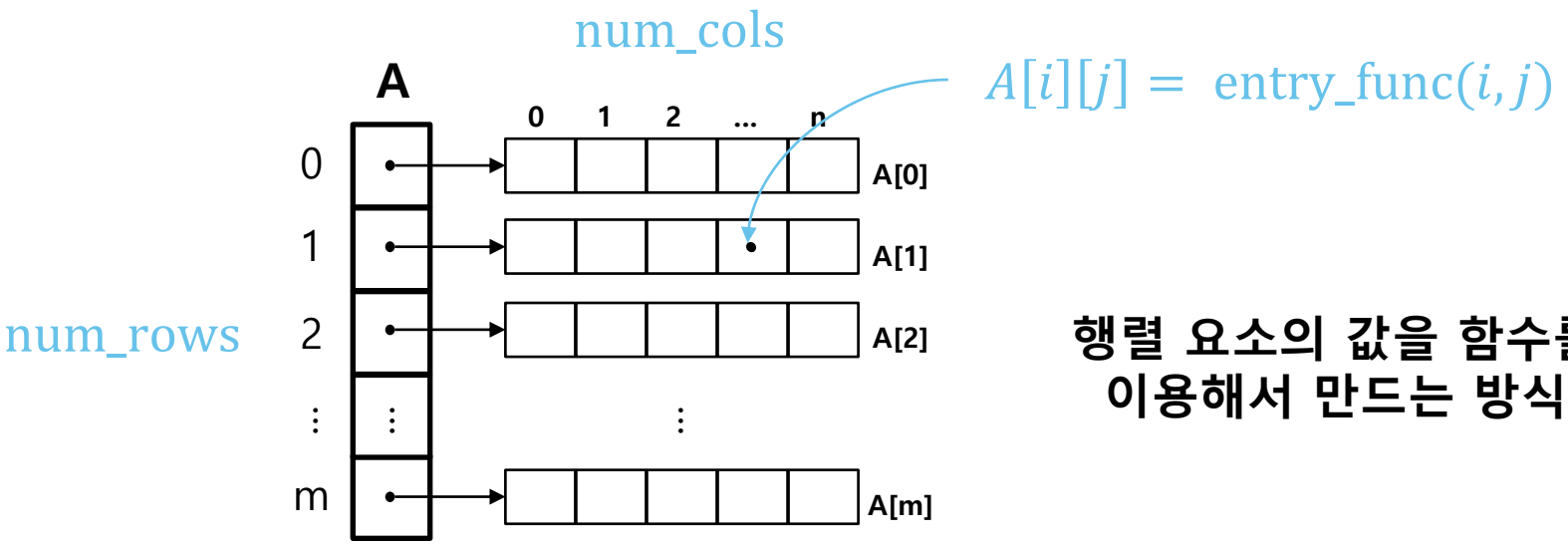
i번째 행

```
def get_row(A: Matrix, i: int) -> Vector:
    """Returns the i-th row of A (as a Vector)"""
    return A[i]             # A[i] is already the ith row
```

j번째 열

```
def get_column(A: Matrix, j: int) -> Vector:
    """Returns the j-th column of A (as a Vector)"""
    return [A_i[j]          # jth element of row A_i
            for A_i in A]   # for each row A_i
```

행렬 만들기

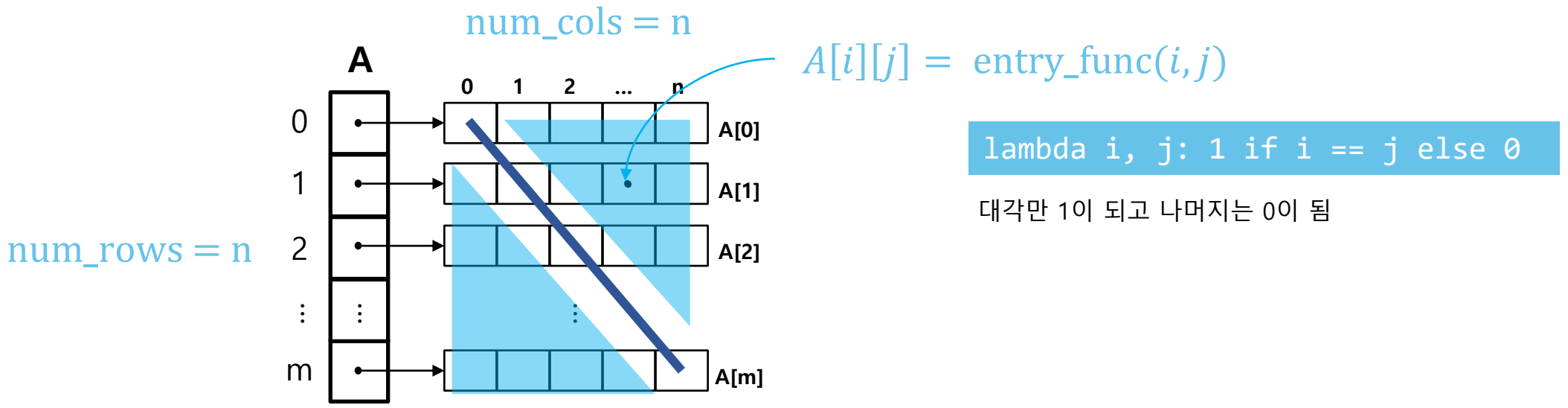


행렬 요소의 값을 함수를 이용해서 만드는 방식

행렬 만들기

```
def make_matrix(num_rows: int,
                num_cols: int,
                entry_fn: Callable[[int, int], float]) -> Matrix:
    """
    Returns a num_rows x num_cols matrix
    whose (i,j)-th entry is entry_fn(i, j)
    """
    return [[entry_fn(i, j)             # given i, create a list
             for j in range(num_cols)]  #   [entry_fn(i, 0), ... ]
            for i in range(num_rows)]   # create one list for each i
```

단위 행렬 만들기




```
lambda i, j: 1 if i == j else 0
```

대각만 1이 되고 나머지는 0이 됨

단위 행렬 만들기

```
def identity_matrix(n: int) -> Matrix:
    """Returns the n x n identity matrix"""
    return make_matrix(n, n, lambda i, j: 1 if i == j else 0)
```

- 행과 열의 크기는 n으로 사각 행렬이 되도록 함
- 대각만 1이 되고 나머지는 0이 되게 하는 람다 함수를 make_matrix에 전달

데이터셋 표현

데이터셋을 표현한다면?

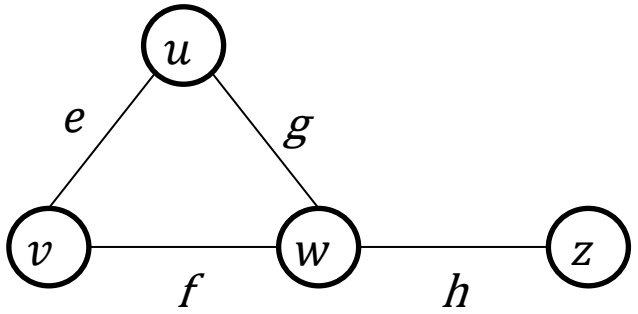
$$A_{m\times n} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \ddots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_{\text{n개의 특징을 표현}} \} \text{m개의 데이터를 표현}$$

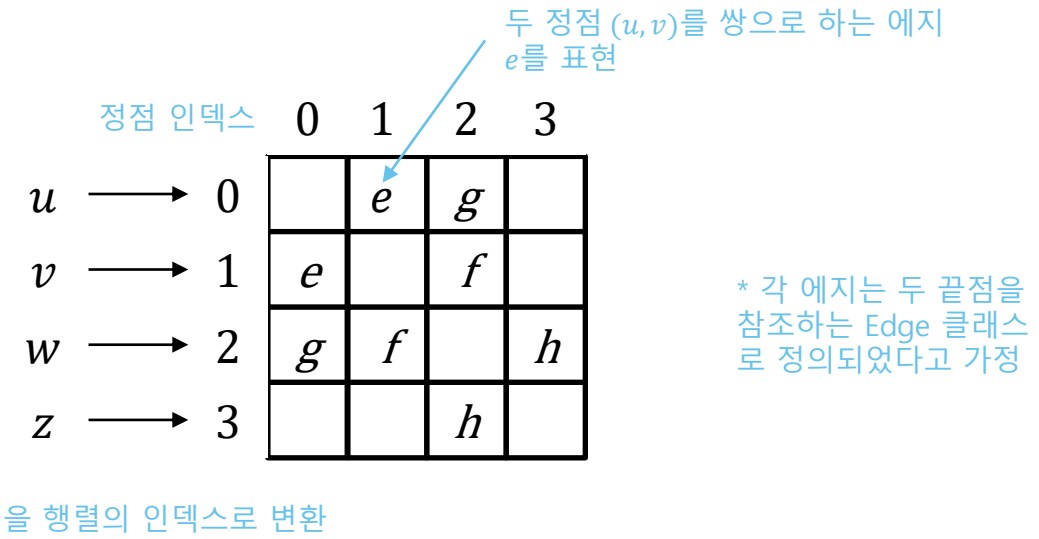
인접 행렬 (Adjacency Matrix)

인접 행렬은 에지 리스트 구조를 행렬로 확장한 구조

두 정점을 쌍으로 하는 에지를 $O(1)$에 접근하기 위해 2차원 행렬로 모든 에지를 표현

그래프 G

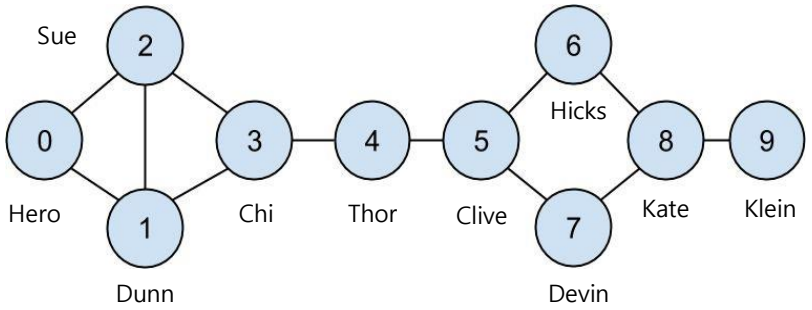




* 각 에지는 두 끝점을 참조하는 Edge 클래스로 정의되었다고 가정

- $A[i,j]$는u의 인덱스가 i이고 v 의 인덱스가 j일 때 에지 (u,v)를 표현
- 무방향 그래프에서는 행렬 A는 항상 대칭(symmetric)임

그래프의 인접 행렬 표현



그래프의 노드 개수가 잘 변하지 않고 에지가 있는지 빠르게 확인해야할 때 인접 행렬을 사용한다.

에지 리스트

```
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

인접행렬

```
friend_matrix = [[0, 1, 1, 0, 0, 0, 0, 0, 0, 0],  # user 0
                 [1, 0, 1, 1, 0, 0, 0, 0, 0, 0],  # user 1
                 [1, 1, 0, 1, 0, 0, 0, 0, 0, 0],  # user 2
                 [0, 1, 1, 0, 1, 0, 0, 0, 0, 0],  # user 3
                 [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],  # user 4
                 [0, 0, 0, 0, 1, 0, 1, 1, 0, 0],  # user 5
                 [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],  # user 6
                 [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],  # user 7
                 [0, 0, 0, 0, 0, 0, 1, 1, 0, 1],  # user 8
                 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]]  # user 9
```

그래프의 인접 행렬 표현

한 사람이 누구와 연결되어 있는지 확인할 때

```
# only need to look at one row
friends_of_five = [i
                   for i, is_friend in enumerate(friend_matrix[5])
                   if is_friend]
```

Thank you!